

# JUQUEEN

## Best Practices

29. November 2013 | Florian Janetzko

# Outline

## Production Environment

- Module Environment
- Job Execution



## Basic Porting

- Compilers and Wrappers
- Compiler/Linker Flags



## Tuning Applications

- Advanced Compiler/Linker Flags
- QPX



# JUQUEEN – System Architecture

## IBM Blue Gene/Q JUQUEEN



- IBM PowerPC A2 1.6 GHz, 16 cores/node, 4-way SMT, 64-bit

- 4-wide (dbl) SIMD (FMA)

- 16 GB RAM per node



- Torus network



- 28 racks, 458,752 cores

- 5.9 Petaflop/s peak



- Connected to a Global Parallel File System (GPFS) with 10 PByte online disk and 37 PByte offline tape capacity



# JUQUEEN – Challenges



## Chip

- 4-way SMT with 1 integer + 1 FPU instruction per cycle → filling pipes
- 4-wide SIMD → efficient vectorization



## Memory

- 1 GB/core and 0.5 GB/core for pure MPI codes → memory consumption
- HW support for transactional memory → efficient usage



## Network

- Torus network → Mapping of tasks, communicators (communication pattern)



## I/O

- Processing large amounts of data → Efficient I/O strategy and management



## Parallelism

- MPP system → Scalability

# Outline

## Production Environment

- Module Environment
- Job Execution



## Basic Porting

- Compilers and Wrappers
- Compiler/Linker Flags

## Tuning Applications

- Advanced Compiler/Linker Flags
- QPX

# Module Environment

## Module concept

- Provides overview over available software packages
- Eases use of software packages
  - Access to software packages, libraries
  - Supply of different versions of applications
  - Supply of application-specific information
- Enables dynamic modification of users' environment
  - Environment variables (`PATH`, `LD_LIBRARY_PATH`, `MANPATH`, ...) are set appropriately
  - Detection of conflicts between applications

# Module Environment

```
$ module <options> <module>
```

Option	Description
<no option>	Lists available options of the module command
avail	Lists all available modules
list	Lists modules currently loaded
load	Loads a module
unload	Unloads a module
help	Lists information about a module
show	Information about settings done by the module
purge	Unloads all modules

# Module Environment

## Six module categories

- **COMPILER**
  - *Different compilers and versions of compilers*
- **IO**
  - *I/O libraries and tools*
- **MATH**
  - *Mathematical libraries and software packages*
- **MISC**
  - *Software not fitting into another category*
- **SCIENTIFIC**
  - *Software packages from different scientific fields*
- **TOOLS**
  - *Performance analysis, debugger, etc.*




### Software for

*Compute Nodes:*

**/bgsys/local**

*Front-end Nodes:*

**/usr/local**





# Module Environment

```
$ module avail
----- /bgsys/local/modulefiles/TOOLS -----
----- /bgsys/local/modulefiles/SCIENTIFIC -----
cp2k/2.2.12394          cpmd/3.15.1_c(default)  namd/2.8
cpmd/3.15.1             lammps/30Aug12          namd/2.9(default)
cpmd/3.15.1_a          lammps/5May12(default)
cpmd/3.15.1_b          libint/1.1.4
```

```
----- /usr/local/modulefiles/COMPILER -----
cmake/2.8.8(default)
----- /usr/local/modulefiles/MATH -----
----- /usr/local/modulefiles/SCIENTIFIC -----
----- /usr/local/modulefiles/IO -----
----- /usr/local/modulefiles/TOOLS -----
UNITE/1.1
----- /usr/local/modulefiles/MISC -----
```

# Module Environment – Applications & Libraries

## Selected mathematical applications and libraries



arpack (2.1)	gsl (1.15)	mumps (4.10.0)	scalapack (2.0.2)
fftw (2.1.5, 3.3.3)	hypre (2.9.0)	parmetis (4.0.2)	sprng (2.0)
gmp (5.0.5)	lapack (3.4.2)	petsc (3.4.2)	sundials (2.5.0)

## Selected scientific applications



CPMD (3.15.3)	Gromacs (4.5.5)	OpenFOAM (2.1.1)
CP2K (2.3.12494)	Lammps (5May12,30Aug12)	QE (5.0.1)
GPAW*	Namd (2.8, 2.9)	VASP**

\* In preparation

\*\* Software not installed but makefiles are available

# Module Environment – Applications & Libraries

## Selected I/O libraries and tools



Darshan (2.2.4)

netcdf (4.3)

SIONlib (1.4.3)

HDF5 (1.8.11)

parallel-netCDF (1.3.1)

## Selected tools



Cmake (2.8.11)

hpctoolkit (5.3.2)

Tau (2.22.3b4)

Clang (3.4)

PAPI (5.1.1)

Totalview (8.12.0)

extrae (2.4.1)

Scalasca (2.1)

Vampir (8.1)

# Outline

## Production Environment

- Module Environment
- Job Execution

## Basic Porting

- Compilers and Wrappers
- Compiler/Linker Flags

## Tuning Applications

- Advanced Compiler/Linker Flags
- QPX



# Running Simulations – Batch System

## Execution of applications managed by LoadLeveler

- Users submit jobs using a job command file
- LoadLeveler allocates computing resources to run jobs
- The scheduling of jobs depends on
  - *Availability of resources*
  - *Job priority (jobs with larger core counts are privileged)*
- Jobs run in queues (job classes)
  - *Classes chosen by LoadLeveler according to core count of job*

For Information about LoadLeveler on JUQUEEN see

<http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUQUEEN/UserInfo/LoadLeveler.html>

# LoadLeveler - Commands

Command	Description
<code>llsubmit &lt;jobfile&gt;</code>	Sends job to the queuing system
<code>llq</code> <code>llq -l &lt;job ID&gt;</code> <code>llq -s &lt;job ID&gt;</code>  <code>llq -u &lt;user&gt;</code>	Lists all queued and running jobs detailed information about the specified job detailed information about a specific queued job, e.g. expected start time lists all jobs of the specified user
<code>llcancel &lt;job ID&gt;</code>	Kills the specified job
<code>llstatus</code>	Displays the status of LoadLeveler
<code>llclass</code>	Lists existing classes and their properties
<code>llqx</code>	Shows detailed information about all jobs

# LoadLeveler – Command Examples

Submitting a batch job: **llsubmit**

```
$ llsubmit batch-job.js
llsubmit: Processed command file through Submit Filter:
"/bgdata/admin/loadl/extensions/filter".
llsubmit: The job "juqueen2c1.zam.kfa-juelich.de.35395" has been submitted.
```

Query status of submitted jobs: **llq**

```
$ llq -u userid
```

Id	Owner	Submitted	ST	PRI	Class	Running On
juqueen2c1.35395.0	userid	11/15 10:11	I	50	n001	

1 job step(s) in query, 1 waiting, 0 pending, 0 running, 0 held, 0 preempted

Cancel a submitted job: **llcancel**

```
$ lllcancel juqueen2c1.35395.0
llcancel: Cancel command has been sent to the central manager.
```

# LoadLeveler – Job Command File

ASCII file containing two major parts

1. LoadLeveler job keywords block at the beginning of a file
  - LoadLeveler keywords have the form  
`#@<keyword>`
  - # and @ can be separated by any number of blanks
2. One or more application script blocks
  - Regular shell script
  - Can contain any shell command



# LoadLeveler – Standard Keywords

Keyword	Description
<code>#@job_name=&lt;name&gt;</code>	Name of the job
<code>#@notification=</code> <code>complete</code> <code>error</code> <code>never</code> <code>start</code> <code>always</code>	Send notification if the job is finished if the job returned an error code $\neq 0$ never upon the start of the job combination of <b>start</b> , <b>end</b> , <b>error</b>
<code>#@notify_use=&lt;mailaddr&gt;</code>	Mail address to send messages to
<code>#@wall_clock_limit=hh:mm:ss</code>	Requested wall time for the job
<code>#@input=&lt;input file name&gt;</code> <code>#@output=&lt;file name for stdout&gt;</code> <code>#@error=&lt;file name for stderr&gt;</code>	Specifies corresponding file names
<code>#@environment=[&lt;variable&gt;, COPY_ALL]</code>	Environment variable to be exported to job
<code>#@queue</code>	Queue job

# LoadLeveler – Blue Gene/Q Keywords

Keyword	Description
<code>#@job_type=[serial, bluegene]</code>	Specifies the type of job step to process. Must be set to <b>bluegene</b> for parallel applications.
<code>#@bg_size=&lt;number of nodes&gt;</code>	Size of the Blue Gene job, keywords <b>bg_size</b> and <b>bg_shape</b> are mutually exclusive.
<code>#@bg_shape=&lt;A&gt;x&lt;B&gt;x&lt;C&gt;x&lt;D&gt;</code>	Specifies the requested shape of a job (in midplanes).
<code>#@bg_rotate=[True,False]</code>	whether the scheduler should consider all possible rotations of the given shape
<code>#@bg_connectivity=[TORUS,MESH,EITHER]</code> <code>Xa Xb Xc Xd</code>	Type of wiring requested for the block (can be specified for each dimension separately)

## LoadLeveler – Job Classes

Class name	#Nodes	Max. run time	Default run time
n001	1 – <b>32</b>	00:30:00	00:30:00
n002	33 – <b>64</b>	00:30:00	00:30:00
n004	65 – <b>128</b>	12:00:00	06:00:00
n008	129 – <b>256</b>	12:00:00	06:00:00
m001	257 – <b>512</b>	24:00:00	06:00:00
m002	513 – <b>1024</b>	24:00:00	06:00:00
m004	1025 – <b>2048</b>	24:00:00	06:00:00
m008	2049 – <b>4096</b>	24:00:00	06:00:00
m016*	4097 – <b>8192</b>	24:00:00	06:00:00
m032*	8193 – <b>16384</b>	24:00:00	06:00:00
m048*	16385 – <b>24576</b>	24:00:00	06:00:00
m056*	24577 – <b>57344</b>	24:00:00	06:00:00


 INCREASING PRIORITY

\*On demand only



**You will be charged for the **full partition** (e.g. if you request 513 nodes you will be charged for **1024** nodes!) → Always use full partitions!**



# LoadLeveler – Job Scheduling

## Backfill scheduler

- The biggest job has the highest priority (*Top Dog*)
- LoadLeveler fills gaps with smaller, short-running jobs while freeing the system for the Top Dog



Tip: Specify the wall time for your jobs as exact as possible, because jobs requesting a shorter wall time have a better chance to be executed.



## Big jobs

- Jobs requesting  $\geq 8$  racks are collected and run in dedicated time slots (e.g. after a maintenance) at least once a week

# Running Simulations – runjob Command

Launch command for parallel applications

```
runjob [options]  
runjob [options]: <executable> [arguments]
```

Option	Description
<code>--args &lt;prg_arg&gt;</code>	Passes " <b>prg_arg</b> " to the launched application on the compute node.
<code>--exe &lt;executable&gt;</code>	Specifies the full path to the executable
<code>--envs &lt;ENV_Var=Value&gt;</code>	Sets the environment variable <b>ENV_Var=Value</b>
<code>--exp-env &lt;ENV_Var&gt;</code>	Sets the environment variable <b>ENV_Var</b>
<code>--np &lt;number&gt;</code>	Total number of (MPI) tasks
<code>--ranks-per-node &lt;number&gt;</code>	Number of (MPI) tasks per compute node

## LoadLeveler – Example Job Command File I

```
#@job_name           = MPI_code
#@comment            = "32 ranks per node"
#@output             = test_$(jobid)_$(stepid).out
#@error              = test_$(jobid)_$(stepid).err
#@environment        = COPY_ALL
#@job_type           = bluegene
#@notification       = never
#@bg_size            = 512
#@bg_connectivity    = torus
#@wall_clock_limit   = 14:00:00
#@queue

runjob --np 16384 --ranks-per-node 32 --exe app.x
```



**Pure MPI applications need to use 32 tasks per node in order use the architecture efficiently!**



# Running Simulations – MPI/OpenMP Codes

- On Blue Gene/P
  - Three modes were available
    1. VN mode (4 MPI tasks, no thread per task)
    2. DUAL mode (2 MPI tasks with 2 OpenMP threads each)
    3. SMP mode (1 MPI task with 4 OpenMP threads)
- On Blue Gene/Q
  - One node has 16 cores with 4-way SMT each
  - Several configurations possible
    - `ntasks × nthreads = 64`
    - `ntasks = 2n, 0 ≤ n ≤ 6`



**Test carefully, which configuration gives the best performance for your application and setup!**



## LoadLeveler – Example Job Command File II

```
#@job_name           = hybrid_code
#@comment            = "16x4 configuration"
#@output             = test_$(jobid)_$(stepid).out
#@error              = test_$(jobid)_$(stepid).err
#@environment        = COPY_ALL
#@job_type           = bluegene
#@notification       = never
#@bg_size            = 512
#@bg_connectivity    = torus
#@wall_clock_limit   = 14:00:00
#@queue

runjob --np 8192 --ranks-per-node 16\
      --env OMP_NUM_THREADS=4 : app.x -i input
```



# Monitoring of Jobs

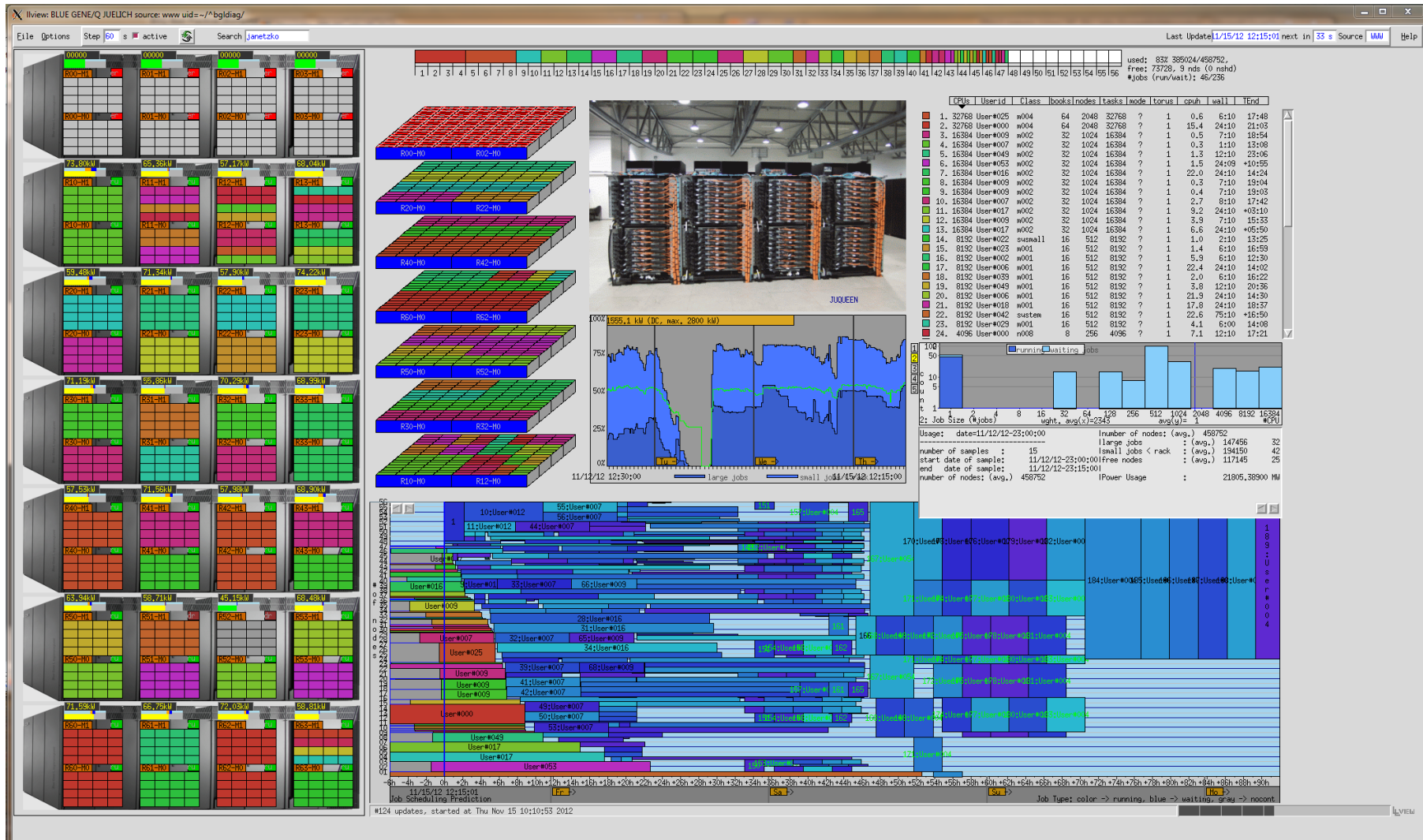
## LoadLeveler

- llq [options]

## Llview

- Client-server based application
- compact summary of different information (e.g. current usage of system, job prediction, expected and average waiting times, ...)
- Customizable
- Developed by W. Frings (JSC)

# Lview



# Outline

## Production Environment

- Module Environment
- Job Execution

## Basic Porting

- Compilers and Wrappers
- Compiler/Linker Flags



## Tuning Applications

- Advanced Compiler/Linker Flags
- QPX

# Compilers

- Different compilers for front-end and compute nodes
- GNU and IBM XL family of compilers available



Tip: It is recommended to use the XL suite of compilers for the CN since they produce in general better optimized code.



FRONT-END	Language	XL compiler	GNU compiler
	C	<code>xlc, xlc_r</code>	<code>gcc</code>
	C++	<code>xlc++, xlc++_r, x1C, x1C_r</code>	<code>g++</code>
	Fortran	<code>xlf, xlf90, xlf95, xlf2003</code> <code>xlf_r, xlf90_r, xlf95_r, xlf2003_r</code>	<code>gfortran</code>

# Compilers for CN

GNU (GCC)	Language	Compiler invocation		MPI wrapper
	C	powerpc64-bgq-linux-gcc		mpigcc
	C++	powerpc64-bgq-linux-g++		mpig++
	Fortran	powerpc64-bgq-linux-gfortran		mpigfortran
XL COMPILERS		(thread-safe: *_r)		
	C	bgxlc, bgc89, bgc99		mpixlc
	C++	bgxlc++, bgxlC		mpixlcxx
	Fortran	bgxlf, bgxlf90, bgxlf95, bgxlf2003		mpixlf77, mpixlf90, mpixlf95, mpixlf2003




**Tip:** To be on the safe side, always use the corresponding MPI wrappers with ‘\_r’ when compiling for the CNs.




# Basic Compiler/Linker Options – XL Compilers I

Flags in order of increasing optimization potential

Optimization Level	Description
-O2 -qarch=qp -qtune=qp	Basic optimization
-O3 -qstrict -qarch=qp -qtune=qp	More aggressive, not impact on acc.
-O3 -qhot -qarch=qp -qtune=qp	More aggressive, may influence acc. (high-order transformations of loops)
-O4 -qarch=qp -qtune=qp	Interprocedural optimization at compile time
-O5 -qarch=qp -qtune=qp	Interprocedural optimization at link time, whole program analysis



**Some flags need to be used during compilation AND linking – Check the compiler manual. If you are not sure, include flags used for compiling also in the linking step!**



# Basic Compiler/Linker Options – XL Compilers II

## Additional compiler flags

Compiler/Linker Flag	Description
<code>-qsmp=omp -qthreaded</code>	Switch on OpenMP support
<code>-qreport -qlist</code>	Generates for each source file <code>&lt;name&gt;</code> a file <code>&lt;name&gt;.lst</code> with pseudo code and a description of the kind of code optimizations which were performed
<code>-qessl -lessl[smp]bg</code>	Compiler attempts to replace some intrinsic FORTRAN 90 procedures by essl routines where it is safe to do so

How to link with ESSL routines, see

[http://www.fz-juelich.de/ias/js/EN/Expertise/Support/Software/SystemDependentLibraries/ESSL\\_ESSLsMPJuqueen.html](http://www.fz-juelich.de/ias/js/EN/Expertise/Support/Software/SystemDependentLibraries/ESSL_ESSLsMPJuqueen.html)

# Outline

## Production Environment

- Module Environment
- Job Execution

## Basic Porting

- Compilers and Wrappers
- Compiler/Linker Flags

## Tuning Applications

- Advanced Compiler/Linker Flags
- QPX





# Diagnostic Compiler Flags (XL Compilers)

Diagnostic messages are given on the terminal and/or in a separate file

- **-qreport**: compilers generate a file **name.lst** for each source file
- **-qlist**: compiler listing including an object listing
- **-qlistopt**: options in effect during compilation included in listing

## Listen to the compiler!

- **-qflag=<listing-severity>:<terminal-severity>**
- i: informal messages, w: warning messages, s: severe errors
- Use **-qflag=i:i** to get all information
- **-qlistfmt=(xml|html)=<option>**

# Example: Compilers Diagnostics

```
subroutine mult(c,a,ndim)
```

```
implicit none
```

```
integer :: ndim,i,j
```

```
double precision ::
```

```
a(ndim),c(ndim,ndim)
```

```
! Loop
```

```
do i=1,1000
```

```
do j=1,1000
```

```
c(i,j) = a(i)
```

```
enddo
```

```
enddo
```

```
end subroutine mult
```

```
>>>> LOOP TRANSFORMATION SECTION <<<<
```

```
1| SUBROUTINE mult (c, a, ndim)
```

```
[...]
```

```
Id=1 DO $$CIV2 = $$CIV2,124
```

```
10| IF (.FALSE.) GOTO lab_11
```

```
$$LoopIV1 = 0
```

```
Id=2 DO $$LoopIV1 = $$LoopIV1,999
```

```
[...]
```

```
-----
0 9 1 Loop interchanging applied
to loop nest.
```

```
0 9 1 Outer loop has been
unrolled 8 time(s).
```



# Single-Core Optimization – Compiler/Linker Flags

## **-qessl**

- For Fortran codes
- If either **-lessl** or **-lesslsmp** are also specified then ESSL routines should be used in place of some Fortran90 intrinsic procedures when there is a safe opportunity to do so.

## **-qipa** (compiling *and* linking step)

- Turns on or customizes interprocedural analysis (IPA)
- High potential for performance benefits
- May *considerably* increase time for compiling and linking step!

# Single-Core Optimization – Compiler/Linker Flags

**-qinline[=auto:level=5, +procedure1[:procedure2[:...]], ...]**

- Attempts to inline procedures instead of generating calls to those procedures, for improved performance.
- Several suboptions are available, check the man page.

**-qsimd=auto**

- Enables the automatic generation of vector instructions

**-qtm**

- Enables support for transactional memory
- Use only with thread-safe compiler wrappers

**-qunroll=yes**

- Instructs the compiler to search for more opportunities for loop unrolling than that performed with auto.

# Outline

## Production Environment

- Module Environment
- Job Execution

## Basic Porting

- Compilers and Wrappers
- Compiler/Linker Flags

## Tuning Applications

- Advanced Compiler/Linker Flags
- QPX

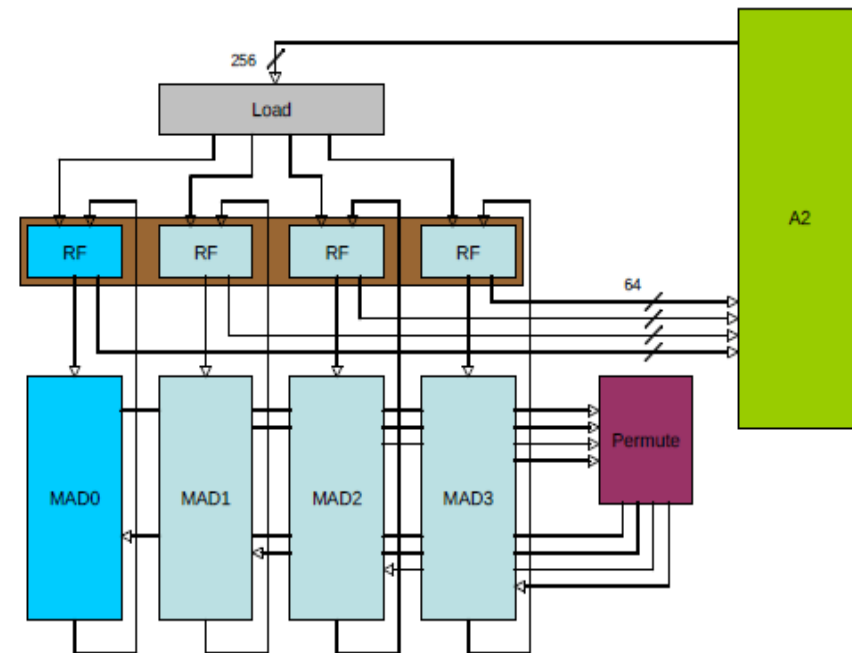


# Quad Floating Point Extension Unit (QPX)

4 double precision pipelines, usable as:

- scalar FPU
- 4-wide FPU SIMD  
(Single Instruction Multiple Data)
- 2-wide complex arithmetic SIMD

8 concurrent floating point ops (FMA) + load + store



# IBM XL Compiler Support for QPX

## Usage of QPX

- Compiler flag `-qsimd=auto`
- **Check** that simd vectorization is actually done!
  - `-qreport`
  - `-qlist`

```
>>>> LOOP TRANSFORMATION SECTION <<<<
[...]
```

```
-----
0 9 1      Loop with nest-level 1 and
           iteration count 1000 was
           SIMD vectorized
```

```
[...]
```

```
>>>> LOOP TRANSFORMATION SECTION <<<<
[...]
```

```
-----
0 9 1      Loop was not SIMD vectorized
           because the loop is not the
           innermost loop.
```

```
0 10 1     Loop was not SIMD vectorized
           because it contains memory
           references with non-
           vectorizable alignment.
```

# QPX Usage – Hints for the Compiler

## Compiler needs hints

- Hint compiler to likely iteration counts
  - Instruct compiler to align fields
  - Tell that FORTRAN assumed-shape arrays are contiguous
- `-qassert=contig`

### Fortran

```
real*8 :: x(:), y(:), a
!ibm* align(32, x, y)
!ibm* assert(itercnt(100))
do i=m, n
    z(i) = x(i) + a*y(i)
enddo
```

### C/C++

```
double __align(32) *x, *y;
double a;
#pragma disjoint(*x, *y)
#pragma disjoint(*x, a)
#pragma ibm iterations(100)
for (int i=m; i<n; i++)
    z[i] = x[i] + a*y[i]
void foo(double* restrict a1,
         double* restrict a2) {
    for (int i=0; i<n; i++) a1[i]=a2[i];
}
```



# IBM XL QPX Ininsics

New intrinsic variable type

- C/C++: `vector4double`
- FORTRAN: `vector(real(8))`

Wide set of elemental functions available

- `LOAD,STORE, MULT, MULT-ADD, ROUND, CEILING, SQRT,...`

Strengths:

- User may layout calculation by hand, if compiler not smart enough (e.g. where no loop)

Easy to use:

- Leave stack, register layout, load/store scheduling to compiler

# QPX Example using Compiler Intrinsics

```
...
typedef vector4double qv;
qv dx,dy,dz,dx2,dy2,dz2
for (i=0;i<4;i++)
{
...
    xd[i] = xdip1[j];
    yd[i] = ydip1[j];
    zd[i] = zdip1[j];
}
dx2 = vec_mul(dx,dx);
dy2 = vec_mul(dy,dy);
dz2 = vec_mul(dz,dz);

d = vec_swsqrt(dx2+dy2+dz2);
...
```

Source: IBM Corporation

# User Information and Support

## Information about JUQUEEN

- JSC websites at  
[http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUQUEEN/JUQUEEN\\_node.html](http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUQUEEN/JUQUEEN_node.html)
- IBM Blue Gene/Q Application Development Redbook  
<http://www.redbooks.ibm.com/redpieces/pdfs/sq247948.pdf>

## Dispatch and User Support

- Applications for accounts (for approved projects)

Forschungszentrum Jülich GmbH, JSC, Dispatch, 52425  
Jülich

Tel: +49 2461 61 5642, Fax: +49 2461 61 2810

email: [dispatch.jsc@fz-juelich.de](mailto:dispatch.jsc@fz-juelich.de)

- User Support

Tel: +49 2461 61 2828

email: [sc@fz-juelich.de](mailto:sc@fz-juelich.de)

Workshop Announcement:

## **Second JUQUEEN Porting and Tuning Workshop**

3-5 February 2014

Forschungszentrum Jülich  
Jülich Supercomputing Centre

<http://www.fz-juelich.de/ias/jsc/events/juqueenpt14>

Contact: Dr. Dirk Brömmel, [d.broemmel\(at\)fz-juelich.de](mailto:d.broemmel@fz-juelich.de)

# END